SDEV 1201 Database Fundamentals

LESSON 5

Let’s review

Query Syntax

Queries are written with the SELECT statement. The simplified syntax for a select statement is:

```
SELECT [ALL | DISTINCT] column_list
[FROM table_name [, table_name2 […, table_name]] -- source of data
          [INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN] -- other sources
[WHERE clause] -- filter
[GROUP BY clause] -- aggregate by
[HAVING clause] -- filter by aggregate
[ORDER BY clause] -- sorts results
```

If it looks complex don’t worry. All the “clauses” listed don’t need to be used when creating a query. However, what clauses you do use MUST be in the order shown above (i.e. HAVING MUST be after GROUP BY for example and not vice versa).

Simple SELECT

```
SELECT
    FirstName
, LastName
, City

FROM Student;
```

Derived fields need a name!

There is another difference with the query that returns the names separately than the one that returned them as one column. The column in the second query does not have a name. This is a problem. We should give all columns a name. Calculated columns do not have a name because they are calculated from many different columns and values. So, we must give the column a name. If the example, the “AS” keyword is optional.

```
Select FirstName || ' ' || LastName AS "Student Name", City
From Student;
```

In this course you are mandated to ALWAYS have column names.

Where

So far, we have only selected data that includes ALL the records on the table. While this is not uncommon, it is likely that you will not only want to select the columns you want, but also only the records you want. The WHERE clause allows you to limit the records that are returned based on criteria you provide.

```
Select StudentID, FirstName, LastName, Gender, StreetAddress, City,
Province, PostalCode, Birthdate, BalanceOwing
From Student
Where StudentID = '198933540';
```

This query would return all the student information for the student with StudentID 198933540. The “Where” clause can contain many different expressions to identify the records the query should return.

Like

The Like operator is used when you use wildcards in your search or perform pattern matching. Wildcards are characters that when used with the Like operator allow the wildcards to represent a character or multiple characters. The Like operator is case sensitive in PostgreSQL.

For example, if we wanted to return all the student's names whose names started with the letter 'D' we would use the % wildcard character. The % wildcard means any character and any number of characters.

```
Select FirstName || ' ' || LastName "Student Name"
From Student
Where FirstName Like 'D%';
```

It is important to use the like operator when using wildcards. The % would be interpreted as a literal '%' if we used = instead of Like.

Like

The _ (underscore) is a single character wildcard. This would be used when you are looking for any one character in your search criteria. If we wanted the students' names whose first name was 3 characters long and started with 'Ja' we could use this query.

```
Select FirstName || ' ' || LastName "Student Name"
From Student
Where FirstName Like 'Ja_';
```

If we had used a % instead of _ we would also return students whose First Name was longer than 3 characters.

Aggregate Functions

Aggregate functions calculate a single value using the data from many records. If you have used function in Excel such as SUM, AVERAGE, COUNT, etc…. then you have used aggregate functions.

```
aggregate_function_name ([ ALL | DISTINCT ] expression)
```

- `AVG(ColumnName)` returns the average of numeric values. `NULL` is ignored.
- `SUM(ColumnName)` returns the sum of a column containing numeric values.
- `MIN(ColumnName)` and `MAX(ColumnName)` returns the minimum & maximum values from a column of numeric, date, or character values.
- `COUNT(*)` returns the number of records that match the `WHERE` criteria.
- `COUNT(ColumnName)` returns the number of non-null values in a given column for records that match the `WHERE` criteria.

Basic Select Exercise 2

Today’s Objective

By the end of this section, you will learn:

- **String and Date Functions**
- **Aggregate** functions and **Group By / Having**.

Full documentation is on the “2C Basic Select Statement” document located in the Brightspace site in the “Week 4 – Select Statements” section.

Assignment Out: Take-home Coding Assignment: Basic SELECT statements

Due September 26, 2025.

String Functions

String Functions

- LENGTH(*column* | *expression*)
- LEFT(*column* | *expression*, *length*)
- RIGHT(*column* | *expression*, *length*)
- SUBSTRING(*column* | *expression*, *start*, *length*)
- REVERSE(*column* | *expression*)
- UPPER(*column* | *expression*)
- LOWER(*column* | *expression*)
- LTRIM(*column* | *expression*)
- RTRIM(*column* | *expression*)

String Functions

LENGTH (column | expression)

```
Select LENGTH (FirstName), FirstName
From Student;
```

Returns the length of each FirstName and the FirstName of each Student.

```
Select LENGTH (FirstName), FirstName
From Student
Where LENGTH (FirstName) = 3;
```

Returns all the First Names that are 3 characters long.

String Functions

LEFT (column |expression, length)

```
Select LEFT (FirstName, 3)
From Student;
```

Returns the first 3 characters from the FirstName of each student.

RIGHT (column |expression, length)

```
Select RIGHT (FirstName, 3)
From Student;
```

Returns the last 3 characters from the FirstName of each student.

String Functions

SUBSTRING (column | expression, start, length)

```
Select SUBSTRING (CourseID, 5, 3)
From Course;
```

In each courseID returns 3 characters starting at the 5th character.

REVERSE (column | expression)

```
Select REVERSE (FirstName)
From Student;
```

Returns the Student first names in reverse.

String Functions

UPPER (column | expression) LOWER (column | expression)

```
Select UPPER (FirstName) "UpperCase", LOWER (LastName) "LowerCase"
From Student;
```

Returns the names in either uppercase or lowercase.

LTRIM (column | expression) RTRIM (column | expression)

```
Select LTRIM (RTRIM ("     Hello World     "));
```

Removes the leading (LTRIM) or trailing (RTRIM) spaces from the column data or expression. To remove leading and trailing spaces you must nest one function in another.

Date Functions

Date Functions

- `Current_Date` returns the current system date
- `Current_Time` returns the current system time
- `DATE_PART(xx, date1)` returns integer representation of the `xx` of `date1`

`xx` represents field for date portions (see next slide).

Field for Date Portions

hour Hour of Day 0 to 23
day Day of Month 1 to 31(depending on month)
dow Day of Week 0 to 6 (Sunday to Saturday)
isodow Day of Week 1 to 7 (ISO numbering)
(Monday to Sunday)
doy Day of Year 1 to 365/366
week Week number of the year 1 to 53
month Month number of the year 1 to 12
quarter Quarter number of the year 1 to 4
year
decade
century
millennium

Field for Time Portions

microseconds

milliseconds

second

minute

Date Function Examples

Find who were hired in the year 2020:

```
select FirstName || ' ' || LastName "Staff Name", DateHired
from Staff
where Date_Part('Year', DateHired) = 2020
```

The number of days between 2 dates:

```
select ('2025-09-17'::date - '2025-09-02'::date) AS difference_in_days;
```

Date Function Examples

-- You can also use Fields to get the character representation of dates.

```
Select To_Char (Current_Date, 'DAY');
```

Returns the day of the week (i.e. MONDAY, TUESDAY, etc.). Note that the "Field" is case sensitive: "Day" would return "Monday".

```
Select To_Char (Current_Date, 'MONTH');
```

Returns the month name. Note that the "Field" is case sensitive: "Month" would return "September".

Group By

The GROUP BY clause is used in conjunction with aggregate functions to provide subtotal calculations of the aggregate function. We could easily return the average Mark from the grade table. What if we wanted the average Mark for each Course? For each student? For each course for each student? To do this we would use GROUP BY.

```
Select AVG(Mark)"Average Mark"
From Registration;
```

Returns the average of all the marks in the Registration table.

Group By

What if we wanted the average mark of each Student?

```
Select CourseID, AVG(Mark) "Average Mark"
From Registration
Group BY CourseID;
```

Returns the average mark for each CourseID (GROUP BY CourseID) in the Registration Table. Another way to think of the syntax GROUP BY is the ‘for each’.

Group By

It is important to note that you cannot combine an aggregate column with a non-aggregate column(s) unless you group by all the non-aggregate columns in your query.

```
Select CourseID, AVG(Mark)"Average Mark"
From Registration;
```

```
ERROR: column "registration.courseid" must appear in the GROUP BY clause or be used in an
 aggregate function
```

Class Activity

Please work on “Basic Select Exercise 3” found in the “Week 4 – SELECT Statements” section of Brightspace.

Having

Having is like the where clause. Except, it applies its criteria after group by has grouped its aggregate data. With the GROUP BY clause, we selected the average mark of CourseID. What if we only wanted to return the CourseIDs and average mark for courses that have averages that are greater than 80?

```
Select CourseID, AVG(Mark)"Average Mark"

From Registration

Group BY CourseID

Where AVG(Mark) > 80;
```

```
ERROR: syntax error at or near "Where“
```

Fails because the where clause must be before the Group By Clause. Remember, a query does not need to use all the clauses but they must be in the right order!

Having

```
Select CourseID, AVG(Mark)"Average Mark"

From Registration

Where AVG(Mark) > 80

Group BY CourseID;
```

```
ERROR: aggregate functions are not allowed in WHERE
```

Fails because and aggregate cannot be in a where clause. It also does not really make sense. It is not possible to return only the CourseID's that have average over 80 until all the Course averages have been calculated. And that is done by the Group By Clause. So, the Group By Clause must execute before the criteria is applied to return only the CourseID's with averages over 80.

Having

```
Select CourseID, AVG(Mark)"Average Mark"
From Registration
Group BY CourseID
Having AVG(Mark) > 80;
```

Works!The Select with the group by selects the averages for each course and then the having clause returns only the ones whose average was over 80.

Having

What if we only wanted to return the CourseID’s that have averages over 80 and did not want the actual average returned. Simply do not select the AVG(Mark). Remember, what we select (return to the user) does not have to match columns used in the where or having clauses.

```
Select CourseID
From Registration
Group BY CourseID
Having AVG(Mark) > 80;
```

Works!

Having

Having is used with aggregate functions only. It is used to apply where clause functionality AFTER the aggregate function and group by has occurred. Only use having in this situation. For queries without a group by clause you should use the WHERE clause.

Homework

Please work on “Basic Select Exercise 4” found in the “Week 4 – SELECT Statements” section of Brightspace. Have it completed before next class.

Important: Please review and finish Basic Select Exercises 1-4 before next class.

Assignment Out: Take-home Coding Assignment: Basic SELECT statements

Due September 26, 2025.